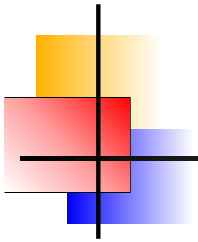
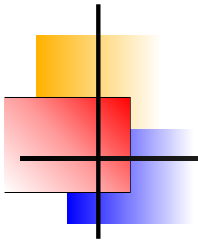


On the Generation of Initial Contexts for Effective Deadlock Detection

Miguel Isabel Márquez
Elvira Albert and Miguel Gómez-Zamalloa

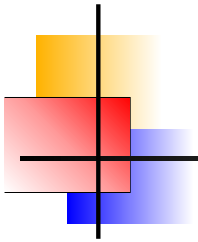
Complutense University of Madrid (Spain)

Namur. October 10, 2017



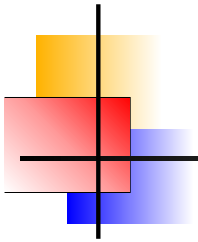
Introduction

- *Testing* is a technique used to ensure validation on programs



Introduction

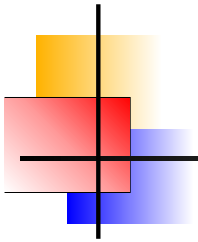
- *Testing* is a technique used to ensure validation on programs
  - *Symbolic execution* performs testing without any information on the input data



Introduction

- *Testing* is a technique used to ensure validation on programs
  - *Symbolic execution* performs testing without any information on the input data

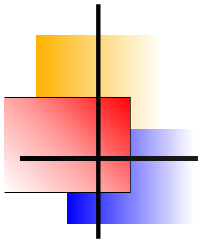
- Deadlock is one of the most common programming errors
- Limitation of *Static Deadlock Analysis*
  - Deadlock analyses can verify absence of deadlock, but when a potential deadlock is detected...
    1. It can be a *false positive*
    2. They provide little information
    3. It can be difficult to find the actual source of the problem



Introduction

- During the symbolic execution...

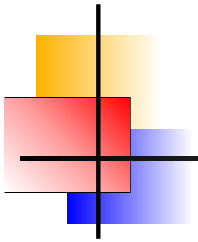
- Problems:



Introduction

- During the symbolic execution...
  - All possible interleavings must be considered
  - The search space may be huge

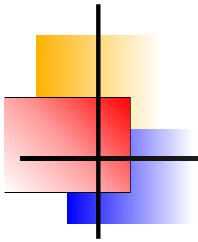
- Problems:
  1. Combinatorial explosion on the state space



Introduction

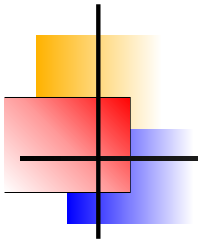
- During the symbolic execution...
  - All possible interleavings must be considered
  - The search space may be huge
  - It needs to start from a concrete initial distributed context

- Problems:
  1. Combinatorial explosion on the state space
  2. Combinatorial explosion on the contexts that must be considered



Introduction

- During the symbolic execution...
  - All possible interleavings must be considered
  - The search space may be huge
  - It needs to start from a concrete initial distributed context
- Problems:
  1. Combinatorial explosion on the state space
  2. Combinatorial explosion on the contexts that must be considered
- We propose to combine deadlock analysis and symbolic execution based testing in order to:
  1. Guide the execution towards paths leading to deadlock

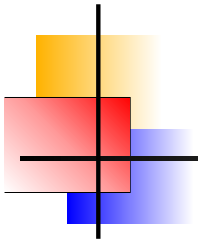


Introduction

- During the symbolic execution...
  - All possible interleavings must be considered
  - The search space may be huge
  - It needs to start from a concrete initial distributed context

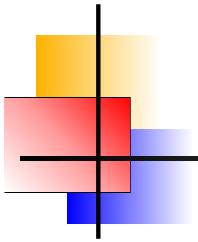
- Problems:
  1. Combinatorial explosion on the state space
  2. Combinatorial explosion on the contexts that must be considered

- We propose to combine deadlock analysis and symbolic execution based testing in order to:
  1. Guide the execution towards paths leading to deadlock
  2. Generate relevant contexts that reduce the number of initial contexts to be executed



Plan of the talk

- Actor-based Concurrency Model
- Generation of Initial Contexts
- Inference of Deadlock-Interfering Tasks
- Conclusions and Future Work



Actor-based Model

- An actor is a monitor that allows at most one active task to execute within the object
  - Tasks are spawned at distributed actors
  - Cooperative scheduling and non-shared memory
    - Asynchronous calls and future variables
  - Operations for *non-blocking* (**await f?**) and *blocking* (**f.get**) synchronization with the termination of tasks

Actor-based Model

Unit main() {

```
  DB db = new DB();
  Worker w = new Worker();
  db!register(w);
  w!work(db);
}
```

class Worker {

```
  Int data;
  Unit work(DB db){
    Fut f;
    f = db!getData(this);
    data = f.get;
  }
  Int ping(Int n){
    return n;
  }
}
```

class DB {

```
  Int data = 1;
  Worker cl = null;
  Unit makesConnection(){
    data = 3;
  }
  Unit register(Worker w){
    data = 1;
    Fut p = this!getData(null);
    await p?;
    if(data>0){
      Fut g = w!ping(5);
      if (g.get == 5)
        cl = w;
    }
  }
  Int getData(Worker w){
    if (cl == w) return data;
    else return null;
  }
}
```

Actor-based Model

Unit main() {

```
→DB db = new DB();
  Worker w = new Worker();
  db!register(w);
  w!work(db);
}
```

class Worker {

```
  Int data;
  Unit work(DB db){
    Fut f;
    f = db!getData(this);
    data = f.get;
  }
  Int ping(Int n){
    return n;
  }
}
```

class DB {

```
  Int data = 1;
  Worker cl = null;
  Unit makesConnection(){
    data = 3;
  }
  Unit register(Worker w){
    data = 1;
    Fut p = this!getData(null);
    await p?;
    if(data>0){
      Fut g = w!ping(5);
      if (g.get == 5)
        cl = w;
    }
  }
  Int getData(Worker w){
    if (cl == w) return data;
    else return null;
  }
}
```

Actor-based Model

Unit main() {

```
  DB db = new DB();
→ Worker w = new Worker();
  db!register(w);
  w!work(db);
}
```

class Worker {

```
  Int data;
  Unit work(DB db){
    Fut f;
    f = db!getData(this);
    data = f.get;
  }
  Int ping(Int n){
    return n;
  }
}
```

class DB {

```
  Int data = 1;
  Worker cl = null;
  Unit makesConnection(){
    data = 3;
  }
  Unit register(Worker w){
    data = 1;
    Fut p = this!getData(null);
    await p?;
    if(data>0){
      Fut g = w!ping(5);
      if (g.get == 5)
        cl = w;
    }
  }
  Int getData(Worker w){
    if (cl == w) return data;
    else return null;
  }
}
```

Actor-based Model

Unit main() {

```
DB db = new DB();
Worker w = new Worker();
db!register(w);
w!work(db);
}
```

class Worker {

```
Int data;
Unit work(DB db){
 Fut f;
 f = db!getData(this);
 data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 1;
  Fut p = this!getData(null);
  await p?;
  if(data>0){
    Fut g = w!ping(5);
    if (g.get == 5)
       cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Actor-based Model

Unit main() {

```
DB db = new DB();
Worker w = new Worker();
db!register(w);
→w!work(db);
}
```

class Worker {

```
Int data;
Unit work(DB db){
  Fut f;
  f = db!getData(this);
  data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 1;
  Fut p = this!getData(null);
  await p?;
  if(data>0){
    Fut g = w!ping(5);
    if (g.get == 5)
      cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Actor-based Model

Unit main() {

```
  DB db = new DB();
  Worker w = new Worker();
  db!register(w);
  w!work(db);
}
```

class Worker {

```
  Int data;
  Unit work(DB db){
    Fut f;
    f = db!getData(this);
    data = f.get;
  }
  Int ping(Int n){
    return n;
  }
}
```

class DB {

```
  Int data = 1;
  Worker cl = null;
  Unit makesConnection(){
    data = 3;
  }
  Unit register(Worker w){
  → data = 1;
    Fut p = this!getData(null);
    await p?;
    if(data>0){
      Fut g = w!ping(5);
      if (g.get == 5)
        cl = w;
    }
  }
  Int getData(Worker w){
    if (cl == w) return data;
    else return null;
  }
}
```

Actor-based Model

Unit main() {

```
DB db = new DB();
Worker w = new Worker();
db!register(w);
w!work(db);
}
```

class Worker {

```
Int data;
Unit work(DB db){
 Fut f;
 f = db!getData(this);
 data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 1;
→ Fut p = this!getData(null);
  await p?;
  if(data>0){
    Fut g = w!ping(5);
    if (g.get == 5)
       cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Actor-based Model

Unit main() {

```
  DB db = new DB();
  Worker w = new Worker();
  db!register(w);
  w!work(db);
}
```

class Worker {

```
  Int data;
  Unit work(DB db){
    Fut f;
    f = db!getData(this);
    data = f.get;
  }
  Int ping(Int n){
    return n;
  }
}
```

class DB {

```
  Int data = 1;
  Worker cl = null;
  Unit makesConnection(){
    data = 3;
  }
  Unit register(Worker w){
    data = 1;
    Fut p = this!getData(null);
    await p?;
    if(data>0){
      Fut g = w!ping(5);
      if (g.get == 5)
        cl = w;
    }
  }
  Int getData(Worker w){
    if (cl == w) return data;
    else return null;
  }
}
```

Actor-based Model

Unit main() {

```
  DB db = new DB();
  Worker w = new Worker();
  db!register(w);
  w!work(db);
}
```

class Worker {

```
  Int data;
  Unit work(DB db){
    Fut f;
    f = db!getData(this);
    data = f.get;
  }
  Int ping(Int n){
    return n;
  }
}
```

class DB {

```
  Int data = 1;
  Worker cl = null;
  Unit makesConnection(){
    data = 3;
  }
  Unit register(Worker w){
    data = 1;
    Fut p = this!getData(null);
    await p?;
    if(data>0){
      Fut g = w!ping(5);
      if (g.get == 5)
        cl = w;
    }
  }
  Int getData(Worker w){
    if (cl == w) return data;
    else return null;
  }
}
```

Actor-based Model

Unit main() {

```
  DB db = new DB();
  Worker w = new Worker();
  db!register(w);
  w!work(db);
}
```

class Worker {

```
  Int data;
  Unit work(DB db){
    Fut f;
    f = db!getData(this);
    data = f.get;
  }
  Int ping(Int n){
    return n;
  }
}
```

class DB {

```
  Int data = 1;
  Worker cl = null;
  Unit makesConnection(){
    data = 3;
  }
  Unit register(Worker w){
    data = 1;
    Fut p = this!getData(null);
    await p?;
→   if(data>0){
      Fut g = w!ping(5);
      if (g.get == 5)
        cl = w;
    }
  }
  Int getData(Worker w){
    if (cl == w) return data;
    else return null;
  }
}
```

Actor-based Model

Unit main() {

```
  DB db = new DB();
  Worker w = new Worker();
  db!register(w);
  w!work(db);
}
```

class Worker {

```
  Int data;
  Unit work(DB db){
    Fut f;
    f = db!getData(this);
    data = f.get;
  }
  Int ping(Int n){
    return n;
  }
}
```

class DB {

```
  Int data = 1;
  Worker cl = null;
  Unit makesConnection(){
    data = 3;
  }
  Unit register(Worker w){
    data = 1;
    Fut p = this!getData(null);
    await p?;
    if(data>0){
→     Fut g = w!ping(5);
      if (g.get == 5)
        cl = w;
    }
  }
  Int getData(Worker w){
    if (cl == w) return data;
    else return null;
  }
}
```

Actor-based Model

Unit main() {

```
DB db = new DB();
Worker w = new Worker();
db!register(w);
w!work(db);
}
```

class Worker {

```
Int data;
Unit work(DB db){
  Fut f;
  f = db!getData(this);
  data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 1;
  Fut p = this!getData(null);
  await p?;
  if(data>0){
    Fut g = w!ping(5);
→   if (g.get == 5)
      cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Actor-based Model

Unit main() {

```
DB db = new DB();
Worker w = new Worker();
db!register(w);
w!work(db);
}
```

class Worker {

```
Int data;
Unit work(DB db){
  Fut f;
  f = db!getData(this);
  data = f.get;
}
Int ping(Int n){
→ return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 1;
  Fut p = this!getData(null);
  await p?;
  if(data>0){
    Fut g = w!ping(5);
    if (g.get == 5)
      cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Actor-based Model

Unit main() {

```
DB db = new DB();
Worker w = new Worker();
db!register(w);
w!work(db);
}
```

class Worker {

```
Int data;
Unit work(DB db){
 Fut f;
 f = db!getData(this);
 data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = w;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 1;
  Fut p = this!getData(null);
  await p?;
  if(data>0){
    Fut g = w!ping(5);
    if (g.get == 5)
→     cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Actor-based Model

Unit main() {

```
DB db = new DB();
Worker w = new Worker();
db!register(w);
w!work(db);
}
```

class Worker {

```
Int data;
Unit work(DB db){
  Fut f;
→ f = db!getData(this);
  data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = w;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 1;
  Fut p = this!getData(null);
  await p?;
  if(data>0){
    Fut g = w!ping(5);
    if (g.get == 5)
      cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Actor-based Model

Unit main() {

```
DB db = new DB();
Worker w = new Worker();
db!register(w);
w!work(db);
}
```

class Worker {

```
Int data;
Unit work(DB db){
 Fut f;
 f = db!getData(this);
→ data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = w;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 1;
  Fut p = this!getData(null);
  await p?;
  if(data>0){
    Fut g = w!ping(5);
    if (g.get == 5)
      cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Actor-based Model

Unit main() {

```
DB db = new DB();
Worker w = new Worker();
db!register(w);
w!work(db);
}
```

class Worker {

```
Int data;
Unit work(DB db){
 Fut f;
 f = db!getData(this);
 data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = w;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 1;
  Fut p = this!getData(null);
  await p?;
  if(data>0){
    Fut g = w!ping(5);
    if (g.get == 5)
      cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Actor-based Model

Unit main() {

```
DB db = new DB();
Worker w = new Worker();
db!register(w);
w!work(db);
}
```

class Worker {

```
Int data = 1;
Unit work(DB db){
 Fut f;
 f = db!getData(this);
 data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = w;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 1;
  Fut p = this!getData(null);
  await p?;
  if(data>0){
    Fut g = w!ping(5);
    if (g.get == 5)
       cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Actor-based Model

Unit main() {

```
DB db = new DB();
Worker w = new Worker();
db!register(w);
w!work(db);
}
```

class Worker {

```
Int data;
Unit work(DB db){
  Fut f;
  f = db!getData(this);
  data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 1;
  Fut p = this!getData(null);
  await p?;
  if(data>0){
    Fut g = w!ping(5);
→   if (g.get == 5)
      cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Actor-based Model

Unit main() {

```
DB db = new DB();
Worker w = new Worker();
db!register(w);
w!work(db);
}
```

class Worker {

```
Int data;
Unit work(DB db){
  Fut f;
→ f = db!getData(this);
  data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 1;
  Fut p = this!getData(null);
  await p?;
  if(data>0){
    Fut g = w!ping(5);
    if (g.get == 5)
      cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Actor-based Model

Unit main() {

```
DB db = new DB();
Worker w = new Worker();
db!register(w);
w!work(db);
}
```

class Worker {

```
Int data;
Unit work(DB db){
  Fut f;
  f = db!getData(this);
→ data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 1;
  Fut p = this!getData(null);
  await p?;
  if(data>0){
    Fut g = w!ping(5);
    if (g.get == 5)
      cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Abstract Deadlock Cycles

Unit main() {

```
DB db = new DB();
Worker w = new Worker();
db!register(w);
w!work(db);
}
```

class Worker {

```
Int data;
Unit work(DB db){
 Fut f;
 f = db!getData(this);
 data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 1;
  Fut p = this!getData(null);
  await p?;
  if(data>0){
    Fut g = w!ping(5);
    if (g.get == 5)
      cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

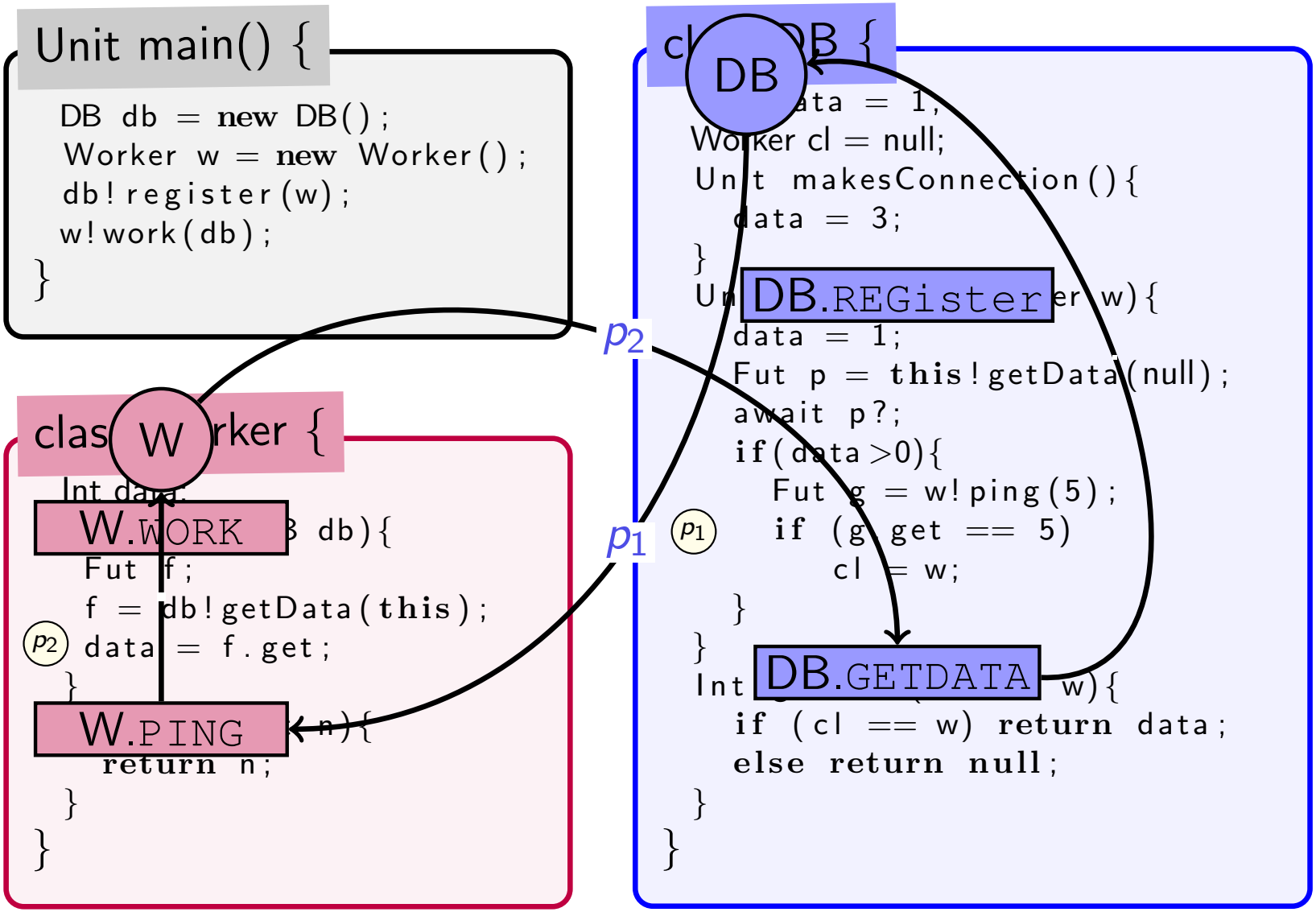
Abstract Deadlock Cycles

```
Unit main() {
  DB db = new DB();
  Worker w = new Worker();
  db!register(w);
  w!work(db);
}
```

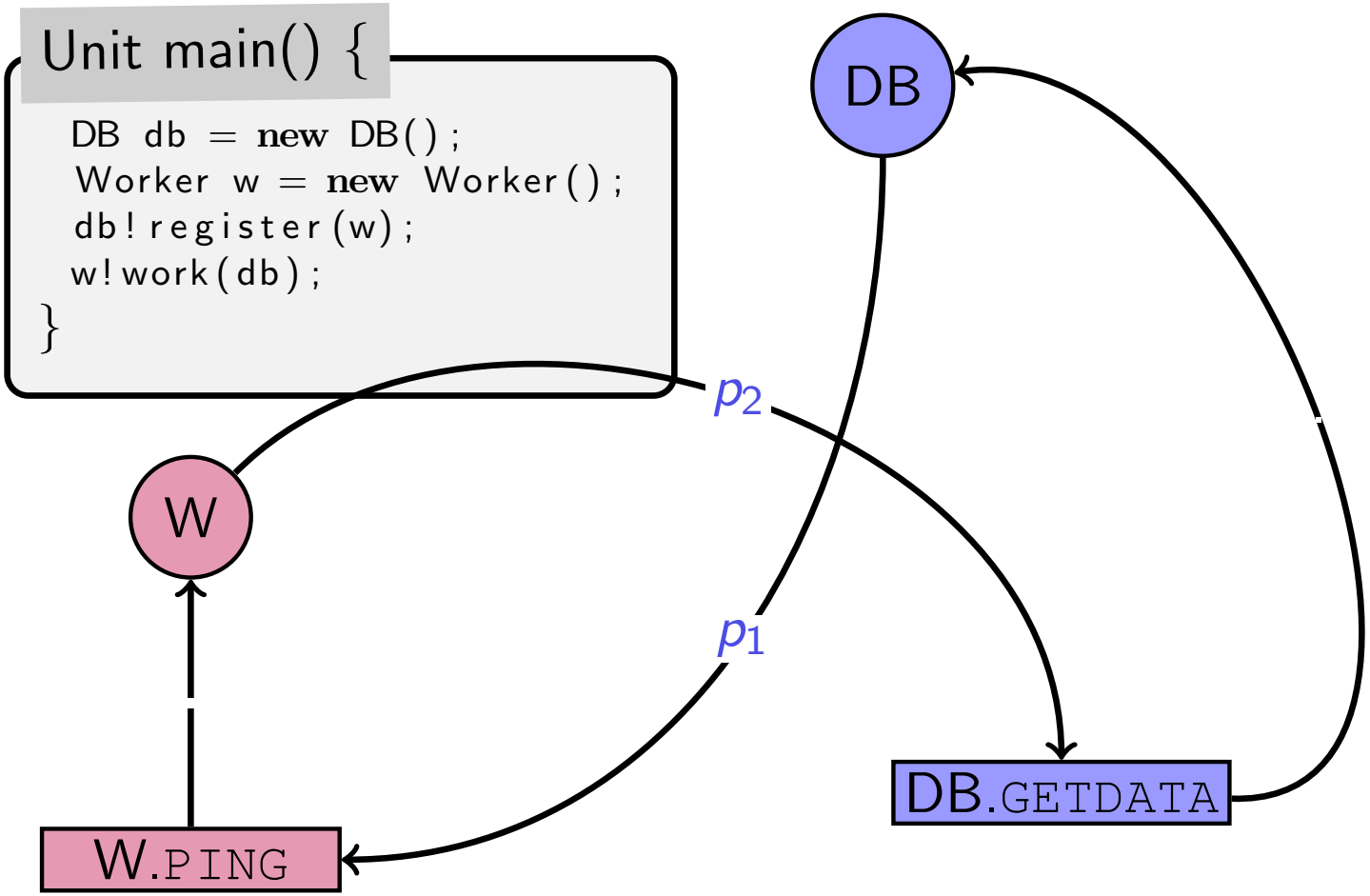
```
class W orker {
  Int data;
  W.WORK  db){
    Fut f;
    f = db!getData(this);
(p2) data = f.get;
  }
  W.PING  n){
    return n;
  }
}
```

```
cl  B {
  DB
  ata = 1;
  Worker cl = null;
  Unit makesConnection(){
    data = 3;
  }
  Un DB.REGister er w){
    data = 1;
    Fut p = this!getData(null);
    await p?;
    if(data>0){
      Fut g = w!ping(5);
(p1)  if (g.get == 5)
        cl = w;
    }
  }
  Int DB.GETDATA  w){
    if (cl == w) return data;
    else return null;
  }
}
```

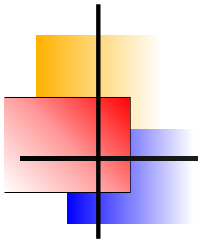
Abstract Deadlock Cycles



Abstract Deadlock Cycles



- Let the abstract deadlock-cycle:
  $c = W \xrightarrow{p_2:WORK} GET \xrightarrow{GET} DB \xrightarrow{p_1:REG} PING \xrightarrow{PING} W$



Extension to the Symbolic Execution

```
Unit main() {
  DB db = new DB();
  Worker w = new Worker();
  db!register(w);
  w!work(db);
}
```

Extension to the Symbolic Execution

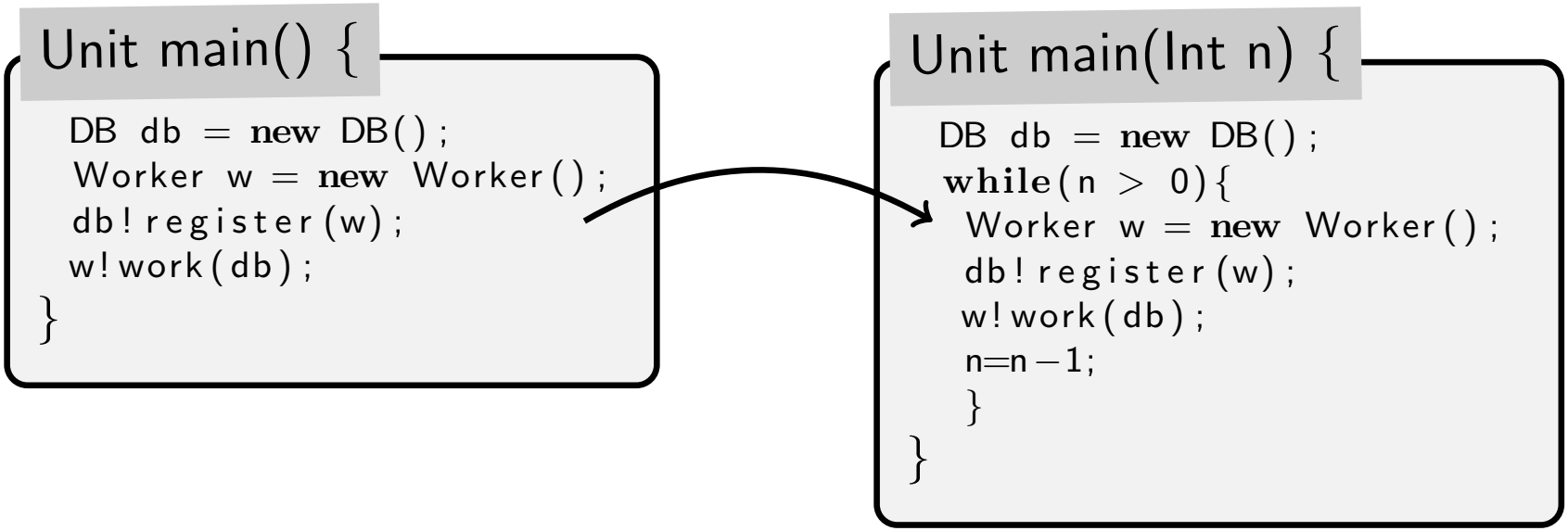
Unit main() {

```
DB db = new DB();
Worker w = new Worker();
db!register(w);
w!work(db);
}
```

Unit main(Int n) {

```
DB db = new DB();
while(n > 0){
 Worker w = new Worker();
 db!register(w);
 w!work(db);
 n=n-1;
 }
}
```

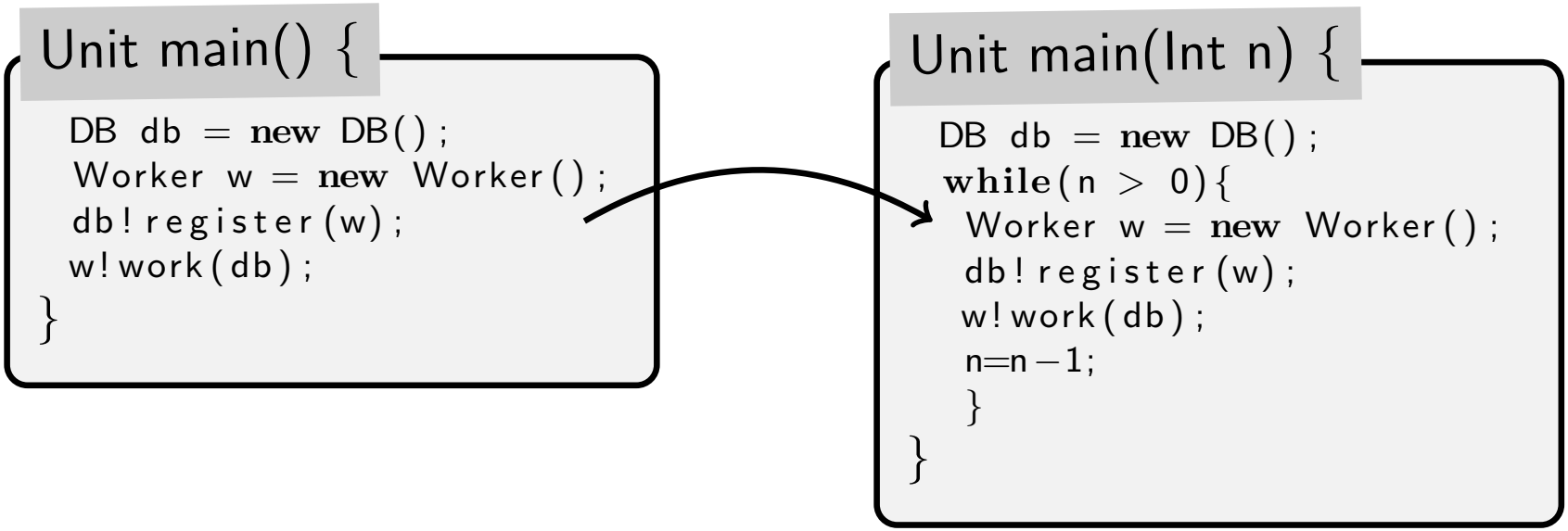
Extension to the Symbolic Execution



Limitations of the symbolic execution:

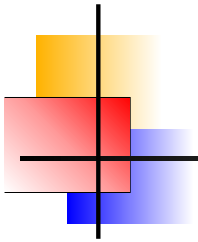
- A limit on the number of loop iterations
- A limit on the number of task switching

Extension to the Symbolic Execution



Limitations of the symbolic execution:

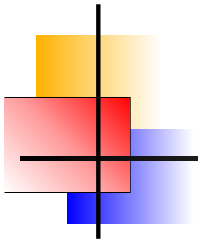
- A limit on the number of loop iterations
- A limit on the number of task switching
- A limit on the number of tasks in the initial context



Generation of initial contexts

- We need to consider a context to start the symbolic execution
- The input to our automatic generation is a set $\mathcal{I}_{ini}$ of tuples

$$(C.M, C^{min}, C^{max})$$



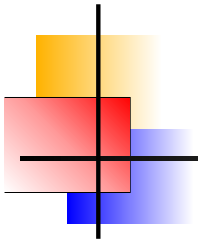
Generation of initial contexts

- We need to consider a context to start the symbolic execution
- The input to our automatic generation is a set $\mathcal{I}_{ini}$ of tuples

$$(C.M, C^{min}, C^{max})$$

- For instance, $\mathcal{I}_{ini} = \{(Worker.work, 1, 1), (DB.register, 1, 1), (DB.makesConnection, 1, 1)\}$, we need to consider the initial contexts:

$\{w_1.work, db_1.makesConnection, db_1.register\}$ and
$\{w_1.work, db_1.makesConnection, db_2.register\}$



Generation of initial contexts

- We need to consider a context to start the symbolic execution
- The input to our automatic generation is a set $\mathcal{I}_{ini}$ of tuples

$$(C.M, C^{min}, C^{max})$$

- For instance, $\mathcal{I}_{ini} = \{(Worker.work, 1, 1), (DB.register, 1, 1), (DB.makesConnection, 1, 1)\}$, we need to consider the initial contexts:

$$\{w_1.work, db_1.makesConnection, db_1.register\} \text{ and}$$
$$\{w_1.work, db_1.makesConnection, db_2.register\}$$

- The systematic generation of initial contexts produces a combinatorial explosion

Inferring Deadlock-Interfering Tasks

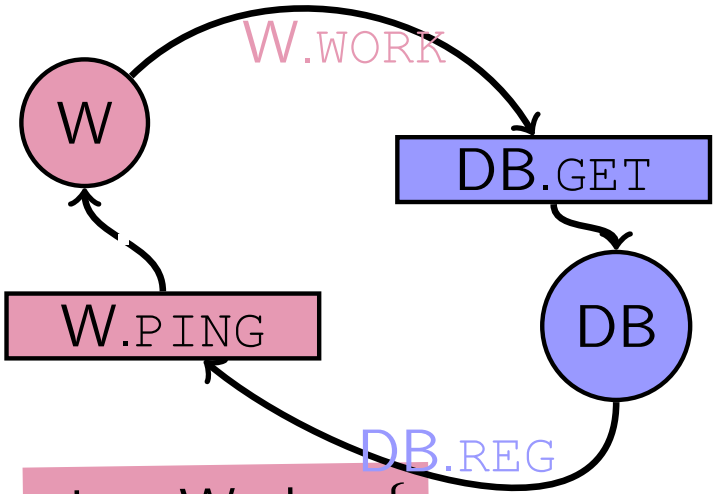
class Worker {

```
Int data;
Unit work(DB db){
 Fut<Data> f;
  f = db ! getData(this);
  data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 0;
  Fut p = this!getData();
  await p?;
   if(data > 0){
      Fut g = w ! ping(5);
       if (g.get == 5) cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Inferring Deadlock-Interfering Tasks



class Worker {

```
Int data;
Unit work(DB db){
 Fut<Data> f;
  f = db ! getData(this);
  data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 0;
  Fut p = this!getData();
  await p?;
   if(data > 0){
      Fut g = w ! ping(5);
       if (g.get == 5) cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Inferring Deadlock-Interfering Tasks

Set of Initial Tasks:

$\mathcal{I}_{ini} =$

class Worker {

```
Int data;
Unit work(DB db){
 Fut<Data> f;
  f = db ! getData(this);
  data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 0;
  Fut p = this!getData();
  await p?;
   if(data > 0){
      Fut g = w ! ping(5);
       if (g.get == 5) cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Inferring Deadlock-Interfering Tasks

Set of Initial Tasks:

$\mathcal{I}_{ini} = \{(\text{Worker.work}, 1, \mathsf{I}_w),$

class Worker {

```
Int data;
Unit work(DB db){
 Fut<Data> f;
  f = db ! getData(this);
  data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 0;
  Fut p = this!getData();
  await p?;
   if(data > 0){
      Fut g = w ! ping(5);
       if (g.get == 5) cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Inferring Deadlock-Interfering Tasks

Set of Initial Tasks:

$\mathcal{I}_{ini} = \{(\text{Worker.work}, 1, \mathsf{I}_w),$

class Worker {

```
Int data;
Unit work(DB db){
 Fut<Data> f;
  f = db ! getData(this);
  data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 0;
  Fut p = this!getData();
  await p?;
   if(data > 0){
      Fut g = w ! ping(5);
       if (g.get == 5) cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Inferring Deadlock-Interfering Tasks

Set of Initial Tasks:
$\mathcal{I}_{ini}$ ={(Worker.work,1,I_w),
(DB.register,1,I_{db}),

class Worker {

```
Int data;
Unit work(DB db){
 Fut<Data> f;
  f = db ! getData(this);
  data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 0;
  Fut p = this!getData();
  await p?;
   if(data > 0){
      Fut g = w ! ping(5);
       if (g.get == 5) cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Inferring Deadlock-Interfering Tasks

Set of Initial Tasks:
$\mathcal{I}_{ini}$ ={(Worker.work,1,I_w),
(DB.register,1,I_{db}),

class Worker {

```
Int data;
Unit work(DB db){
 Fut<Data> f;
  f = db ! getData(this);
  data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 0;
  Fut p = this!getData();
  await p?;
   if(data > 0){
      Fut g = w ! ping(5);
       if (g.get == 5) cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Inferring Deadlock-Interfering Tasks

Set of Initial Tasks:
$\mathcal{I}_{ini}$ ={(Worker.work,1,I_w),
(DB.register,1,I_{db}),
(DB.makesConnection,1,I_{db})}

class Worker {

```
Int data;
Unit work(DB db){
 Fut<Data> f;
  f = db ! getData(this);
  data = f.get;
}
Int ping(Int n){
  return n;
}
}
```

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 0;
  Fut p = this!getData();
  await p?;
   if(data > 0){
      Fut g = w ! ping(5);
       if (g.get == 5) cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Inferring Deadlock-Interfering Tasks

Set of Initial Tasks:
$\mathcal{I}_{ini}$ ={(Worker.work,1,I_w),
(DB.register,1,I_{db}),
(DB.makesConnection,1,I_{db})}

Initial Context 1

{ db_1.*register*
db_2.*makesConnection*
w_1.*work* }

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 0;
  Fut p = this!getData();
  await p?;
   if(data > 0){
      Fut g = w ! ping(5);
       if (g.get == 5) cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```

Inferring Deadlock-Interfering Tasks

Set of Initial Tasks:
$\mathcal{I}_{ini}$ ={(Worker.work,1,I_w),
(DB.register,1,I_{db}),
(DB.makesConnection,1,I_{db})}

Initial Context 1

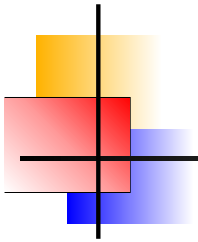
{ db_1.*register*
db_2.*makesConnection*
w_1.*work* }

Initial Context 2

{ db_1.*register*
db_1.*makesConnection*
w_1.*work* }

class DB {

```
Int data = 1;
Worker cl = null;
Unit makesConnection(){
  data = 3;
}
Unit register(Worker w){
  data = 0;
  Fut p = this!getData();
  await p?;
   if(data > 0){
      Fut g = w ! ping(5);
       if (g.get == 5) cl = w;
  }
}
Int getData(Worker w){
  if (cl == w) return data;
  else return null;
}
}
```



Conclusions & Future Work

- Conclusions
  - Framework for effective deadlock detection avoiding the combinatorial explosion problem
  - Useful for the programmers to find deadlocks and fix them
  - The tool `aPET` is available online at `costa.ls.fi.upm.es/syco`
- Future Work
  - Experimental evaluation on generating initial contexts leading to deadlock
  - Improvements on *Deadlock-Interfering Tasks Algorithm*
  - Precision improvements and extensions $\rightarrow$ more precise analyses and more accurate information